MWSUG 2017 - Paper RF06

Cleaning Messy Data: SAS® Techniques to Homogenize Tax Payment Data

Aaron Barker, Iowa Department of Revenue, Des Moines, IA

ABSTRACT

One challenge frequently encountered by analysts aggregating over large volumes of data is dealing with inconsistencies among data sources. Fortunately, all versions of SAS® have a wide array of tools available that a user of any skill level can employ to remedy such problems. This paper uses example tax payment data to demonstrate tools which have been employed to do a novel analysis of citizens' experience with the tax system in Iowa. The first method highlighted is conditional logic on identifier variables to determine how best to interpret the data in the identifier column. This is shown both using IF THEN logic and then again using SELECT and WHEN statements. The second method is determining when identifiers are of different types (e.g. permit numbers vs. Social Security Numbers) and then bringing in external data sources to recode inconsistent identifiers into consistent ones. This is shown using both PROC SQL as well as the traditional MERGE statement. The end result of these procedures is a data set that allows the analyst to see how many times a given individual or business interacts with the Department and provides valuable insight into the preferred payment method of taxpayers by tax type, payment type, and frequency of payments.

INTRODUCTION

Every analyst worth their salt has had to deal with messy data. Some of it is missing, a share is invalid, and one of these rows is not like the others. Cleaning up data makes analysis possible, and there are a wide range of tools in SAS to deal with both common and uncommon problems. This paper uses example tax payment data to present a handful of helpful data cleansing techniques that facilitated a novel analysis of taxpayers' interaction with the Department.

LITERATURE REVIEW

Data cleaning is a popular topic among analysts, and a considerable body of literature already exists on this topic. Cody (2008) is the preeminent source for techniques to improve data quality, including how to use the PROC FREQ procedure to identify invalid character values as well as the handy VERIFY and NOTDIGIT functions. Widawski (2006) highlights the key steps involved in cleaning data: discovering the problem, locating erroneous entries, and fixing the problem. More recently, Chaudhary and Screiber-Gregory (2015) provide some excellent advice in identifying outliers with the PROC UNIVARIATE, PROC MEANS, and PROC SGPLOT procedures and also give a useful list of functions commonly employed in data cleaning (some of which are used later in this paper).

However, this paper differs from previous writings in that most techniques are intended to find input errors that are more or less random; the procedures discussed here were instead developed to take batches of data that are systematically miscoded and try to format them in a similar fashion to enable analysis. In reality, these problems are not "errors" in isolation: they are the unfortunate result of having data management systems that do not align well, which is discussed further in the next section.

SHORT PRIMER ON TAX DATA

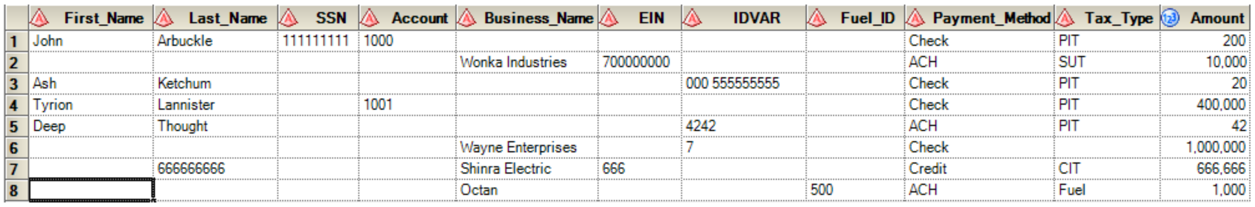
Before turning to the data, it is helpful to understand a little about tax administration. Departments handling tax revenue must deal with a multitude of different types of taxes coming in on various schedules through varying payment methods. The public is most familiar with income taxes, which are generally paid in the spring through the Automated Clearing House (ACH), debit card, credit card, and paper transactions. However, the majority of income tax liability is remitted by employers on behalf of their employees' through withholding payments, which are generally submitted electronically on regular intervals throughout the year. Retailers and other businesses remit sales taxes, which can occur either annually or throughout the year. Citizens often pay property taxes or car registration payments to their local jurisdictions, and those entities often remit some share of the payment to the state. Insurers pay insurance premium taxes, tobacco retailers pay cigarette and tobacco taxes, and alcohol stores pay beer and liquor taxes to just name a few of the many different revenue streams that are present in most states.

To handle this wide array of activity, revenue departments have varying numbers of systems collecting each type of payment or tax. For example, in Iowa, one contractor handles all tax payments made with credit cards, another company and system is used to handle all cigarette tax transactions, the Department's website handles many but not all types of electronic transactions, while delinquent tax payments are handled through yet another system. Each of these systems require the taxpayer to provide different information, often including Social Security Numbers, account numbers, permit numbers, and/or the period for which the payment is due. Each system has the information needed to deposit, track, and administer each tax type independently.

Recently, the Department was asked to determine how often individual and business taxpayers make a payment to the State. The multiplicity of revenue streams and systems mentioned above, unfortunately, make this analysis quite taxing for one salient reason: inconsistent identifiers. In order to complete this data analysis, all of the information from all the different systems noted above was deposited in one dataset on an SQL server. To allow for meaningful reporting, the data set had to be homogenized around a single point of reference. Enter SAS-facilitated data cleansing.

EXAMPLE OF MESSY DATA

With a basic understanding of the situation that generates messy data, it is helpful to have an example SAS dataset with which to work. Some (fictitious) taxpayer data below is presented as Figure 1.



	First_Name	Last_Name	SSN	Account	Business_Name	EIN	IDVAR	Fuel_ID	Payment_Method	Tax_Type	Amount
1	John	Arbuckle	111111111	1000					Check	PIT	200
2					Wonka Industries	700000000			ACH	SUT	10,000
3	Ash	Ketchum					000 555555555		Check	PIT	20
4	Tyrion	Lannister		1001					Check	PIT	400,000
5	Deep	Thought					4242		ACH	PIT	42
6					Wayne Enterprises		7		Check		1,000,000
7		666666666			Shinra Electric	666			Credit	CIT	666,666
8					Octan			500	ACH	Fuel	1,000

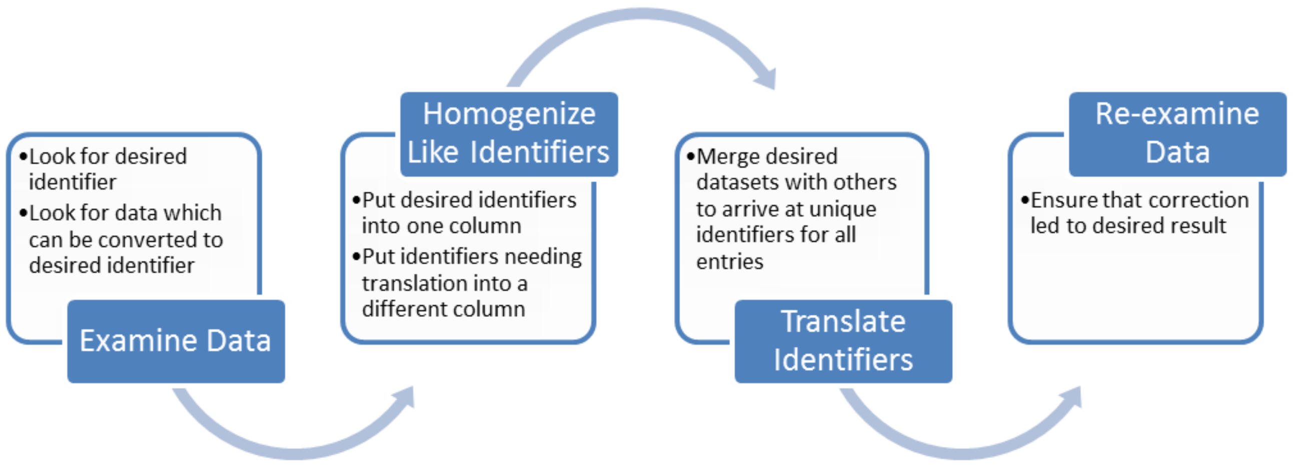
Figure 1. Fictitious Taxpayer Data - work.messy

This data set has a lot of variables to interpret. There are variables for first name and last name, which are self-explanatory. There is also an SSN column, which stands for Social Security Number. If correct, this will be the best identifier of individuals. There's an account number, which is a four digit number used by the collections unit for delinquent payments. The business name column is obvious, and this is followed by an Employer Identification Number which is used to uniquely identify businesses and is thus the best variable for identifying them. This is followed by an IDVAR column, and you have no idea from where this data came (any analyst of someone else's data has run into this). The last identifying variable is FuelID, which is a permit number used by payers of motor fuel tax. The last three variables are how the payment came in (Check, ACH, Credit, etc.), the tax type of the liability that the taxpayer is paying, and the amount that they paid. All variables except amount are character variables, as applying mathematical operations on them is not meaningful.

The objective of this exercise is to develop a data cleansing program to distill out a column of unique identifiers that can be used to determine how many times individual taxpayers, whether people or businesses, interacted with the Department. You will save this in a new column called ID_Number.

FINDING IDENTIFIERS

Before actually getting into the data, it is helpful to have a brief roadmap on what you are about to do. In non-SAS terms, the logic process in Figure 2 shows essentially what you hope to do:



Examine Data
•Look for desired identifier
•Look for data which can be converted to desired identifier
Homogenize Like Identifiers
•Put desired identifiers into one column
•Put identifiers needing translation into a different column
Translate Identifiers
•Merge desired datasets with others to arrive at unique identifiers for all entries
Re-examine Data
•Ensure that correction led to desired result

Figure 2. Logic Chart of Data Cleaning Process

To start, you will examine your data to determine in which column are the identifiers of interest. At the same time, you can also look for non-unique identifiers that you will be able to translate into unique identifiers in the future. After this, you implement SAS coding to bring all of the unique identifiers together in one column; at the same time, you can ensure that non-unique identifying information is categorized consistently. Next, you take this non-unique identifying information and merge it with another data set to translate non-unique identifiers into unique ones. Finally, you should always reexamine your data to ensure that the result is one you expected. You do not want to wind up with an SSN that starts with an H.

METHOD #1: USING CONDITIONAL LOGIC

This first entry looks great. You know John Arbuckle's SSN, his account number, which tax he paid (personal income tax, or PIT), how he paid it, and how much he paid. Since you know John's SSN, you can simply edit your cleansing program to put SSN into ID_Number:

```
data cleansed;
  set messy;
    ID_Number = SSN;
run;
```

The next entry looks pretty good too, but the ID_Number column will not get populated because SSN is blank. This entry is for sales and use tax, or SUT, which businesses collect on behalf of the consumer and remit to the state. So, you can conclude that this is a business taxpayer. Fortunately, you have the EIN of Wonka Industries, so you can use that as the unique identifier. To do that, you need to improve your program to use SSN if it is not blank and then EIN if it is not blank using conditional logic, in this case IF THEN statements. The business line is second because there are probably more individuals than businesses paying taxes. Your code now looks like:

```
data cleansed;
  set messy;
    if SSN NOT = " " then ID_Number = SSN;
    else if EIN NOT = " " then ID_Number = EIN;
run;
```

You run into your first real problem in the next entry, which is missing both an SSN and an EIN. When you look at what information you have, though, you see that there are three zeroes followed by a nine digit number in the mysterious IDVAR column. Whatever made this column looks like it dumped some errata followed by an SSN, so go ahead and treat it as such. To do this, you add another ELSE statement combined with a SUBSTR() statement to pull out the SSN, specifying the target variable (IDVAR), the position you want to start (5), and the number of digits that you want to extract (9). Your code is now:

```
data cleansed;
  set messy;
    if SSN NOT = " " then ID_Number = SSN;
    else if EIN NOT = " " then ID_Number = EIN;
    else if IDVAR NOT = " " then ID_Number = substr(IDVAR,5,9);
run;
```

The next line has nothing in the SSN, EIN, or ID_Number variable; however, you do have an account number. Since you know that account numbers are four digits and this one appears to match that standard, then you presume that this account number is correct. You will translate this into a unique identifier once you get to the PROC SQL and DATA step examples. You do not need to update your code right now.

However, the next line throws a monkey-wrench into the existing program. There's a number in IDVAR, but it looks like a billing account number. So, you will need additional conditional logic to prevent dumping this information into ID_Number and instead put it in the account number variable, which (like the above) you will handle later. In this case, you can use a LENGTH() statement to check how many characters are in IDVAR and then process it accordingly:

```
data cleansed;
  set messy;
    if SSN NOT = " " then ID_Number = SSN;
    else if EIN NOT = " " then ID_Number = EIN;
    else if length(IDVAR) = 13 then ID_Number = substr(IDVAR,5,9);
    else if length(IDVAR) = 4 then Account = IDVAR;
run;
```

Now, you have unique ID_Numbers for the first three entries and account numbers for the next two, which you will soon translate into unique identifiers.

Then, you come to the sixth entry, for which you do not have an SSN, EIN, or anything in the IDVAR. You do not even have a tax type for what liability Wayne Enterprises is paying! This happens surprisingly often in tax administration: taxpayers submit payments with no identifying method for actually attributing the payment to any specific tax liability. In fact, the number of taxpayers who submit a payment without even providing their name or address is shockingly high. Unfortunately, there is no SAS remedy for this dearth of information; you will need to leave it in the data set and treat it as a payment without an identifier.

METHOD #2: USING SELECT AND WHEN STATEMENTS

Using SELECT and WHEN statements are handy when your data has one column that instructs you how to interpret the rest of the line. In this example, you could probably use Tax Type to translate some of the data into unique identifiers.

For example, this alternative code to the above conditional logic takes two large categories of taxpayers and finds their unique identifiers with minimal coding. The OTHERWISE statement is necessary to specify what happens when something in Tax_Type does not meet your given conditions, and the END completes the syntax. In this case, OTHERWISE should be blank because you do not want SAS to change anything unless Tax_Type meets the values we specify. Your alternative code looks like:

```
data cleansed_alt;
  set messy;
    select (Tax_Type);
      when ("PIT") ID_Number = SSN;
      when ("CIT") ID_Number = EIN;
      otherwise;
      end;
run;
```

This could also be employed when you know of some systematic data problem that is dumping information into the wrong column. For example, the seventh entry is the first credit card payment in the data set: when you look at it, you see that the EIN for Shinra Electric is clearly wrong as it is only three digits, and you notice that there is a nine digit number in the Last_Name column, which should be blank. It looks like EIN got dumped into the Last_Name column in the credit card data payment management system, so you can use a SELECT statement to find your identifier; you want to implement this fix before the other SELECT statement because you do not want the erroneous EIN to be put in as an ID_Number. Your revised code would look something like this:

```
data cleansed_alt;
  set messy;
    select (Payment_Method);
      when ("Credit") EIN = Last_Name;
      otherwise;
      end;
    select (Tax_Type);
        when ("PIT") ID_Number = SSN;
    when ("CIT") ID_Number = EIN;
        otherwise;
       end;
run;
```

(It is important to note that this program will put everything in the Last_Name column into the EIN column, so it should only be employed if we are sure that every credit card payment has a similar error.)

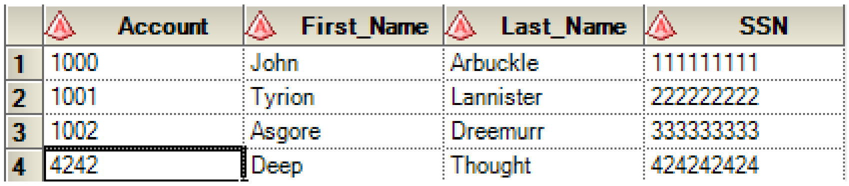
As compared to conditional logic, this program looks a lot cleaner and is very similar to CASE-WHEN statements in SQL. However, it is not nearly as flexible as IF THEN statements, and given the degree to which the starting data set is muddled, you might want to opt for IF THEN logic in this instance

TRANSLATING IDENTIFIERS

METHOD #1: USING PROC SQL

While the above procedures have helped us find identifiers among the data we already have, you often need to find identifying information from another data set. You can use the PROC SQL procedure to quickly find this data.

Before, you observed that the fourth entry in this data has no SSN or EIN but it does have an account number. If you also know that you have another data set that maps account numbers to SSNs, then you can use PROC SQL to merge these two data sets and pull out the useful information. The accounts_db data set containing account numbers and unique identifiers is shown below as Figure 3:



	Account	First_Name	Last_Name	SSN
1	1000	John	Arbuckle	111111111
2	1001	Tyrion	Lannister	222222222
3	1002	Asgore	Dreemurr	333333333
4	4242	Deep	Thought	424242424

Figure 3. Fictitious Accounts Data - work.accounts_db

You use a left join in this instance because you do not want to lose any of the rows in the original data set that do not have an account number. For those not particularly familiar with PROC SQL, the “t1” and “t2” become aliases for data sets, and variables within each are preceded by these aliases and a dot. The two data sets are merged based on the variables specified in parentheses, which is account in this case. The SSN variable from the second data set is renamed SSN2 to indicate that this came from a different source than the original SSN.

The PROC SQL procedure is followed by a DATA step to put the pulled SSNs into the ID_Number variable. (Note that this code uses the conditional logic result [work.cleansed] instead of the SELECT WHEN result [work.cleansed_alt], as the former result is cleaner.) The code which you will use to translate your identifiers looks like:

```
proc sql;
  create table cleansed2 as select
    t1.*,
    t2.SSN as SSN2
  from cleansed t1
    left join accounts_db t2 on (t1.account = t2.account);
quit;
data cleansed3 (drop=SSN2);
  set cleansed2;
  if ID_Number = “ ” and SSN2 NOT = “ ” then ID_Number = SSN2;
run;
```

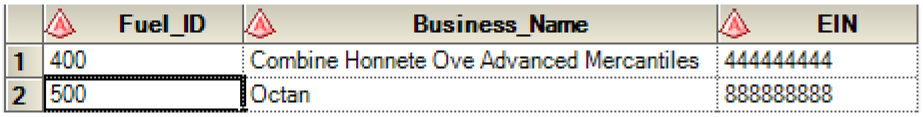
It should be noted that right now, the above code is set up to translate account numbers to unique identifiers after doing some initial data cleaning. The reason for this is that you discovered account numbers hidden in other variables, such as IDVAR, so that you can translate all of the entries that have account numbers but are missing ID_Numbers in one step. Otherwise you might end up doing this data translation twice.

METHOD #2: USING DATA STEP MERGE

Using a MERGE statement in a data step is another way to bring in another data set to make sense of your original. While the DATA step is a little more intuitive to code than PROC SQL, the restriction that entries have to be sorted by the merged variable makes the code a little longer.

The eighth entry has a FuelID number but no permit or EIN. While it is possible that this is the only tax that Octan pays, it is likely that the company also has some corporate income or withholding tax liability. Thus, if you know that you have another data set that has FuelIDs and EINs, then you can bring that in to improve your data.

Here is the data for the fuel data set in Figure 4:



	Fuel_ID	Business_Name	EIN
1	400	Combine Honnete Ove Advanced Mercantiles	444444444
2	500	Octan	888888888

Figure 4. Fictitious Fuel Permit Data - work.fuel

Below is the code that you can use to translate these Fuel_IDs into unique identifiers. Here, the matching variable is put in the BY statement. In addition, to ensure that you have only the results that appear in your payments data set (cleansed3), then you must specify (in=a) for the first data set and then add a sub-setting IF statement to get only those entries which appear in cleansed3:

```
proc sort data=cleansed3;
  by Fuel_ID;
run;
proc sort data=fuel out=motorfuel;
  by Fuel_ID;
run;
data cleansed4 (drop=EIN2);
  merge cleansed3(in=a) motorfuel(rename=(EIN=EIN2));
  by Fuel_ID;
  if a;
  if ID_Number = " " and EIN2 NOT = " " then ID_Number = EIN2;
run;
```

This program results in basically the same output as the PROC SQL output above, and you do not need to specify the variables that you want to keep if you do not wish to do so. Of course, if your data files have more than the identifiers you need, then it is possible that you will bring over loads of information that is not relevant to your analysis. In which case, employ DROP and KEEP data step commands to get just what you are after.

PERILS OF FINDING UNIQUE IDENTIFIERS

If this process looks time-consuming, messy, and error-prone, it is because it is. Sorting through various datasets that have not had consistent controls on data integrity is one of the most maddening things that you can do as an analyst. If a data set is being fed from many different systems or entered by people who handle data in an inconsistent fashion, then you will often need to use many SAS procedures to make the data consistent. The code to develop is dependent on the varying systems that are generating your data, so your solution will probably be unique for each file.

Someone with astute attention to data quality concerns might also point out that by putting SSNs and EINs into the same column, you are liable to incorrectly match individuals to businesses where these identifiers are the same. Unfortunately, there is no ready solution to this problem. However, within each state, SSNs often have similar first three digits while FEINs often have similar first two digits, so the pool of individuals and businesses to which to accidentally match (especially in a sparsely-populated state like Iowa) is fairly small.

APPLICATION IN PRACTICE: TAX PAYMENT FREQUENCY PER TAXPAYER

In the State of Iowa, techniques like these were used to create a cleansed data set that listed an ID_Number for over 94 percent of tax payments. By counting and summing amounts by ID_Number and tax type or payment type, the Department discovered a whole lot of interesting, previously unknowable facts about Iowa taxpayers. This data was further augmented by bringing in date of birth and whether or not the taxpayer used a professional tax preparer to prepare and/or file their return. For example, we learned the following for FY 2016:

- 52.8% of taxpayers made only one payment.
- 86.4% of taxpayers made five or fewer payments.
- 0.03% of taxpayers made 79 or more payments (entirely businesses).
- Individual estimate payments are the largest source of paper payments (27.2% of check volume).

- For individual income tax final payments, returns filed by professional tax preparers represent the largest source of paper checks (75% of matched payments and returns).
- 63% of people under 25 and 61% of people between ages 25-34 pay via check.
- 53 taxpayers paid through MoneyGram.

CONCLUSION

This paper has employed various techniques for cleaning and managing data to end up with a readable, meaningful data set on which analyses can be performed. This began with IF THEN conditional logic to create a unique identifier column, and then this was performed again using SELECT and WHEN statements. After these techniques got out a lot of the snags, PROC SQL and the DATA step MERGE statement were used to merge the partially-cleaned data set with other existing data sets to arrive at unique identifiers for the vast majority of tax payments. Finally, it presented a sample of findings from the Department's analysis of Iowa tax data from FY 2016 which provide information about the current nature of tax payments. These findings were used to prioritize the Department's technological projects and to understand where best to intervene to encourage electronic payments wherever possible. SAS provides an excellent toolset to prepare data for analysis, and it is hoped that future projects by the Department will provide more nuance and accuracy to the results developed to date.

REFERENCES

Cody, R. 2008. *Cody's Data Cleaning Techniques*. 2nd ed. Cary, NC: SAS Institute.

Chaudhary, K. and D. Schreiber-Gregory. 2015. "Let SAS Cleanse your Dirty Data®." *Proceedings of the Midwest SAS Users Group 2017 Conference*. Omaha, NE: Midwest SAS Users Group.

Widawski, M. 2006. "Clean Up Your Act: Data Cleaning with SAS®." Accessed August 21, 2017. http://www.lexjansen.com/wuss/2006/tutorials/TUT-Widawski.pdf.

ACKNOWLEDGMENTS

This paper would not have been possible without the assistance of two mentors. First is my supervisor Amy Harris who provided encouragement and editing in the drafting process. Her attention to detail never, truly never, ceases to amaze me. The second is Kirk Lafler who provided guidance throughout this process and was delightfully enthusiastic about helping me. I thank both of them very much.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Aaron Barker
(515) 725-4018
Aaron.Barker@iowa.gov